|  |  |
| --- | --- |
|  | **УТВЕРЖДАЮ**  Профессор кафедры  ИАНИ ННГУ, д.т.н.  \_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Старостин  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г. |

РУКОВОДСТВО СИСТЕМНОГО ПРОГРАММИСТА (РСП)

**НИР «Разработка и реализация программного обеспечения   
для решения задачи многомерной аппроксимации функции»**

**(Шифр ПО "APPROX")**

Ответственный исполнитель

\_\_\_\_\_\_\_\_\_\_\_\_\_К.В. Лобанкина

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Оглавление

[Общие сведения о программном обеспечении 3](#_Toc28075906)

[Структура программного обеспечения 4](#_Toc28075907)

[Классы вне пакетов 4](#_Toc28075908)

[Классы пакета *model* 5](#_Toc28075909)

[Классы пакета *solver* 6](#_Toc28075910)

[Классы пакета *classifier* 8](#_Toc28075911)

[Перечень принятых сокращений 10](#_Toc28075912)

# Общие сведения о программном обеспечении

Итоговым продуктом научно-исследовательской работы является программная библиотека, используемая для автоматизации решения задачи многомерной аппроксимации функции, а также консольное приложение, демонстрирующее работу с ним. Данное API имеет возможность разбивать множество входных точек на домены (строить диаграмму Вороного), строить граф соседствующих точек, аппроксимировать функцию на всем множестве точек (аппроксимация Шепарда), оценивать точность текущего приближения (вычислять ошибку для каждого домена), обрабатывать данные исследования и находить прогнозные точки и формировать выходные данные (координаты прогнозных точек).

Данное API представляет собой несколько пакетов классов: model, solver, classifier, а также классы Approx, TestFunctionGetter и Tools. Подробнее компоненты описаны в следующей главе. Библиотека использует вспомогательную библиотеку alglib.

API реализовано на языке С# и может быть беспрепятственно использовано при разработке на этом языке.

# Структура программного обеспечения

## Классы вне пакетов

**class Approx**

Основной класс, содержащий метод Main(string[] args), запускающий работу программы

Поля:

Названия функций, которые могут быть использованы

const string PYRAMYDE\_VOLUME = "PyramidVolume";

const string SIN\_FROM\_SUM\_ON\_SUM = "SinFromSumOnSum";

const string SIN\_X\_COS\_X\_COS\_Y = "SzinXCosXCosY";

const string SIN\_X\_COS\_Y = "SinXCosY";

const string SQUARE\_AREA = "SquareArea";

const string SQUARE\_PRODUCTS = "SquaresProducts";

const string TRUNC\_PYRAMID = "TruncPyramid";

const string LG\_FUNC = "LGFunc";

const string LN\_XY3 = "LnXY3";

const string SQRT\_X\_SQRT\_Y = "SqrtXSqrtY";

Путь до конфигурационного файла

public static string configFile;

Путь до файла, сожержащего непосредственно точки

public static string pointFile;

Массив, содержащий конкретные названия функций, на которых проводится обучение (названия из констант, упомянутых выше)

public static string[] learnFunctions;

Переменная, содержащая конкретное название тестируемой функции (названия из констант, упомянутых выше)

public static string testFunction;

**public class TestFunctionGetter**

Класс для получения конкретных функций из списка выше по имени

Методы:

public static TestFunctionGetter getInstance(string testName)

Возвращает функцию связанную с testName

public Func<double[], double[]> GetFunc()

Возвращает производную связанную с testName

public Func<double[], double[]> GetDerivative()

Устанавливает таблицу для функций, которым это необходимо

public void SetTable(Dictionary<double[], double[]> table)

Классы реализующие конкретные функции:

public static class PyramidVolume

public static class SinFromSumOnSum

public static class SzinXCosXCosY

public static class SinXCosY

public static class SquareArea

public static class SquaresProducts

public static class TruncPyramid

public static class LGFunc

public static class LnXY3

public static class SqrtXSqrtY

**public static class Tools**

Класс реализующий вспомогательные функции

## Классы пакета *model*

**public class Grid**

Класс отвечающий за работу с сетью

Поля:

int N, NM;

double[] min;

double[] max;

public readonly double[] step;

public readonly int[] count;

public readonly int[] rount;

public readonly double[][] Node;

Методы:

public Grid(int N, int M, double[] min, double[] max, int[] count);

public bool ToIndex(double[] x, out int index);

public bool ToIndex(int[] counter, out int index);

public bool ToCounter(int index, int[] counter);

public IEnumerable<int> Neighbours(int index);

**public interface IFunction**

Интерфейс, являющийся основой функции аппроксимации

Свойства:

Количество независимых переменных

int N;

Количество зависимых переменных

int M;

Минимальные значения для функции

double[] Min;

Максимальные значения для функции

double[] Max;

Методы:

По первым N значениям вектора xy вычисляет оставшиеся M значений вектора xy

void Calculate(double[] xf);

**public abstract class AFunction** : IFunction

Частично реализует интерфейс IFunction

**public class Function** : IFunction

Класс, реализующий интерфейс IFunction

**public interface IParser**

Интерфейс, декларирующий поведение парсера

Методы:

Считывает конкретные точки для задачи, из файла с точками, используя готовый объект с конфигурационной информацией config

Task parseTask(string pathToTask, IConfig config);

Считывает конфигурационные данные из файла с конфигурацией

IConfig parseConfig(string pathToConfig);

Считывает изначальные значения для функции, если для нее необходима таблица (то есть зависимость не может быть реализована)

Dictionary<double[], double[]> parseTable(string pathToTable, IConfig config);

**public class Parser** : IParser

Класс реализующий интерфейс IParser

**public class Task**

Основной класс, формирующий задачу

Поля:

Текущие точки, массив double[pointAmount][N+M]

public double[][] points;

Количество независимых переменных в функции

public int N;

Количество зависимых переменных в функции

public int M;

Тестируемая функция

public Func<double[], double[]> function;

Методы:

public Task(double[][] points, int N)

public Task(int pointAmount, int N, int M)

## Классы пакета *solver*

**public interface IAnalyzer**

Интерфейс, декларируюзий поведение анализатора

Методы:

Проведение анализа для регрессора

void do\_random\_forest\_analyse(IClassifierML rg, Func<double[], IFunction, Grid, double[], double[][], int, double[]> build\_features);

Проведение анализа для классификатора

void do\_random\_forest\_analyse(IClassifierML cls, double allowErr, Func<double[], double> meFunc, Func<double[], double[]> calcDerivative);

Свойство для получение результата

double[][] Result { get; }

**public class Analyzer** : IAnalyzer

Реализует интерфейс IAnalyzer

**public interface IConfig**

Интерфейс, декларирующий поведение объекта, содержащего конфигурационные настройки

Свойства:

Количество точек которые было расчитано ранее и подается на вход

int PointAmount { get; set; }

Точность

float Approximation { get; set; }

N - размерность функции

int FunctionDimension { get; set; }

M - число зависимых переменных

int DependentVariablesNum { get; set; }

Количество точек которое мы готовы получить на выходе, чтобы расчитать в них точные значения

int PredictionPointAmount { get; set; }

N-мерный вектор минимальных значений входных парамеров

double[] Min { get; set; }

N-мерный вектор максимальных значений входных парамеров

double[] Max { get; set; }

**public class Parser** : IParser

Реализует интерфейc IParser

**public interface IApprox**

Интерфейс, декларирующий действия аппроксиматора

Методы:

Подсчитывает значение на данных точках

void Calculate(double[] xf);

Подсчитывает ошибку

double GetError(double[] oldPoint, double[] newPoint);

**class ShepardApprox** : AFunction, IApprox

Реализует интерфейсы IApprox и расширяет абстрактный класс AFunction, реализует алгоритм аппроксимации Шепарда

**public interface ISolver**

Основной интерфейс, вызываемый напрямую из функции Main и реализующий основной функционал

void setConfig(IConfig config);

void setCLassifier(Func<double[], double[]>[] functions, Task task);

void setRegressor(Func<double[], double[]>[] functions, Task task);

double calculate(Task task);

void testResult(Func<double[], double[]> func);

void testResult(int nFunctionDimension, int mFunctionDimension, Func<double[], double[]> func);

**public class Solver** : ISolver

Основной класс, реализующий интерфейс ISolver

## Классы пакета *classifier*

**public struct LabeledData**

Структура для работы с размеченными данными

Поля:

public double[] data;

public Object label;

Методы:

public LabeledData(double[] data, int label)

**public abstract class AMLAlgorithmParams**

Абстракнтый класс для параметров для классификатора или регрессора

Поля:

public LabeledData[] XY;

Методы:

public AMLAlgorithmParams(LabeledData[] xy)

**public interface IClassifierML**

Интерфейс декларирующий работу классификатора или регрессора

Методы:

void train<T>(AMLAlgorithmParams param);

void infer(double[] x, out Object label);

void validate<T>(LabeledData[] XY, out double modelPrecision) where T : IComparable;

**public class RandomForest** : IClassifierML

Класс, реализующий интерфейс IClassifierML с помощью схемы Random forest

Поля:

Параметры для классификатора

RandomForestParams param;

**public class RandomForestParams** : AMLAlgorithmParams

Класс, расширяющий класс параметров для классификатора или регрессора

Поля:

Размер сета для тренировки

public int NPoints

Количество независимых переменных

public int NVars

Количество классов

public int NClasses

Количетсво деревьев в forest

Рекомендуемое количество: 50-100.

public int NTrees

Количество переменных используемых для выбора лучшего

public int NRndvars

Процент от тренировочного сета используемых для построения индивидуального дерева 0<R<=1. Рекомендуемые значения: 0.1 <= R <= 0.66.

public double R

Методы:

public RandomForestParams(LabeledData[] xy, int npoints, int nvars, int nclasses, int ntrees, int nrndvars, double r) : base(xy)

# Перечень принятых сокращений

ПО – программное обеспечение

API - Application Programming Interface